实验 20_uCos

【实验目的】:

进行 uC/OS 操作系统在 FS_11C14 开发板上的移植。学习 uC/OS 操作系统与 FS_11C14 平台的综合实践。

【实验环境】:

1、FS_11C14 开发板

2、FS_Colink　V2.0

3、RealView MDK（Keil uVision4）

【实验步骤】:

1、 在“20_uCOS 实验”文件夹下找到并打开 project.uvproj 文件；

2、 编译此工程；

3、 通过 FS_Colink 下载编译好的工程到 FS_11C14 开发板；

4、 按 Reset 键复位开发板；

5、 通过按下按键，打开串口，观察实验现象；

6、 查看相关芯片手册，学习其原理及使用方法；

7、 对照原理图分析实验代码

【实验现象说明】

（1）将程序下载到 FS_11C14 开发板之后，OLED 屏没有显示。数码管进行计数任务，两个 LED 灯在系统时钟中断的控制下进行闪烁；

（2）按下选择按键，会发现风扇转动、蜂鸣器名叫；

（3）再次按下按键，两者停止工作；

（4）打开串口终端，重启开发板，会看到串口中打印相关信息。

【实验程序说明】

1、键盘任务

键盘任务的主要工作就是不断地扫描键盘。

在任务中，创建了任务轮转所需要使用的系统资源：一个互斥信号量。

键盘任务的代码清单如下：

```
void Task_Key(void *pdata)
{
    while(1)
    {
        if((KEY_Read()==KEY_SEL) || (KEY_Read()==KEY_ESC))
        {
            OSSemPost(Key_Sem);              //释放信号量
```

```
        }
        OSTimeDlyHMSM(0, 0, 0, 300);          //延时
    }
}
```

可以看到程序周期性的进行查看，每当开发板上的“SEL”或“ESC”按键按下时，会导致信号量 Key_Sem 的改变，系统资源释放。

2、蜂鸣器、风扇任务

该任务主要实现了对蜂鸣器以及风扇模块的控制。通过键盘任务对信号量的操作来获得信号量，然后完成蜂鸣器和风扇的开关。

程序代码如下：

```
void Task_BeepFan(void *pdata)
{
    uint8_t       on_off=0;
    uint8_t       err;
    while(1)
    {
        OSSemPend(Key_Sem, 0, &err);

        on_off = ~on_off;

        if(on_off == 0)
        {
            GPIOSetValue(PORT0, 2, 0);        // 开风扇
            GPIOSetValue(PORT1, 1, 1);        // 关蜂鸣器
        }
        else
        {
            GPIOSetValue(PORT0, 2, 1);        // 关风扇
            GPIOSetValue(PORT1, 1, 0);        // 打开蜂鸣器
        }
    }
```

```
}
```

每当按下“SEL”或者“ESC”按键时，会完成上面的操作之一，再次按下时，会执行相反的操作。

3、LED灯任务

关于LED灯的控制我们在前面已经介绍过了，只需要通过GPIO口我们就可以实现LED灯的开关。每隔0.5s将两个LED灯的状态反转一次。任务通过消息邮箱完成与其他任务之间的通信。每隔1s计数值加一，从而也实现了7段数码管的周期性的显示。

具体实现如下：

```
void Task_Led(void *pdata)
{
    uint32_t cnt=0;

    while(1)
    {
        GPIOSetValue(PORT3, 0, 0);        // 开 LED1
        GPIOSetValue(PORT3, 1, 0);        // 开 LED2

        // 广播方式发送计数值指针(Mbox)
        OSMboxPostOpt(Led_Mbox, (void *)&cnt,
OS_POST_OPT_BROADCAST);
        cnt ++;
        OSTimeDlyHMSM(0, 0, 0, 500);
        GPIOSetValue(PORT3, 0, 1);        // 关 LED1
        GPIOSetValue(PORT3, 1, 1);        // 关 LED2
        OSTimeDlyHMSM(0, 0, 0, 500);
    }
}
```

4、7段数码管任务

该任务与LED灯任务之间通过消息邮箱进行通信，通过LED任务释放资源，然后该任务接收资源并完成7段数码管的周期显示。具体实现：

```
void Task_Seg7Led(void *pdata)
{
```

```
    uint8_t         err;
    uint32_t *led_cnt;

    while(1)
    {
        led_cnt = (uint32_t *)OSMboxPend(Led_Mbox, 0, &err);
        Seg7Led_Put((*led_cnt)%10);
    }
}
```

在任务获取到资源之后，通过 Seg7Led_Put()函数，将变量 led_cnt 的值在 7 段数码管中得以显示。

5、统计任务

该任务周期的进行各个任务对 CPU 的占用情况以及堆栈的使用情况。我们可以通过串口得到结果。

```
void Task_Monitor(void *pdata)
{
    OS_STK_DATA         stk;
    uint32_t        lu0;

    while(1)
    {
        lu0 = OSTimeGet() / OS_TICKS_PER_SEC;
        printf("\r\n\r\nSecond:  %d,  MCU-Used:  %d%c",  lu0,
OSCPUUsage, '%');

        printf("\r\n----------------------");

        OSTaskStkChk(OS_LOWEST_PRIO-1, &stk);
        printf("\r\nStat:      %4u -%4u =%4u",
                            stk.OSUsed/4+stk.OSFree/4,  stk.OSUsed/4,
stk.OSFree/4);
```

```
        OSTaskStkChk(OS_LOWEST_PRIO, &stk);
        printf("\r\nIdle:       %4u -%4u =%4u",
                              stk.OSUsed/4+stk.OSFree/4, stk.OSUsed/4,
stk.OSFree/4);

        OSTaskStkChk(Task_Led_PRIO, &stk);
        printf("\r\nLed:        %4u -%4u =%4u",
                              stk.OSUsed/4+stk.OSFree/4, stk.OSUsed/4,
stk.OSFree/4);

        OSTaskStkChk(Task_Seg7Led_PRIO, &stk);
        printf("\r\nSeg7Led: %4u -%4u =%4u",
                              stk.OSUsed/4+stk.OSFree/4, stk.OSUsed/4,
stk.OSFree/4);

        OSTaskStkChk(Task_Key_PRIO, &stk);
        printf("\r\nKey:        %4u -%4u =%4u",
                              stk.OSUsed/4+stk.OSFree/4, stk.OSUsed/4,
stk.OSFree/4);

        OSTaskStkChk(Task_BeepFan_PRIO, &stk);
        printf("\r\nBeepFan: %4u -%4u =%4u",
                              stk.OSUsed/4+stk.OSFree/4, stk.OSUsed/4,
stk.OSFree/4);

        OSTaskStkChk(Task_Monitor_PRIO, &stk);
        printf("\r\nMonitor: %4u -%4u =%4u",
                              stk.OSUsed/4+stk.OSFree/4, stk.OSUsed/4,
stk.OSFree/4);

        OSTimeDlyHMSM(0, 0, 1, 0);
```

```
    }

}
```